README file
NIST DSR Model
7 December 2000

CONTENTS[1]

Introduction

The goal of this report is to present and describe the Dynamic Source Routing (DSR) OPNET process model that has been developed at NIST. In this way it will be possible for every OPNET user or developer to use this process model in his simulations.

Thus the first part is devoted to a general overview of the DSR protocol, and of the Mobile Ad Hoc Network (MANET) for which it has been developed.

Then, since the process model cannot be provided without a node model, a presentation of this node model is done in the second part of the report.

Finally, the third and last part of the report focuses on the DSR process model; that is the description of this model comparing to its specification and the presentation of its state machine. Detailed "technical" information relative to the model, such as the variables, the functions, and the packet format descriptions, are available in the appendix.

[1] An appendix containing details of the DSR model implementation is given in a separate file.

General Overview

What is a Mobile Ad Hoc Network?

It is important to raise this question since the Dynamic Source Routing protocol was designed especially for this kind of network.

A Mobile Ad Hoc Network, also called a MANET, is an autonomous collection of mobile nodes forming a dynamic wireless network. The administration of such a network is decentralized, *i.e.* each node acts both as host and router and forwards packets for nodes that are not within transmission range of each other. A MANET provides a practical way to rapidly build a decentralized communication network in areas where there is no existing infrastructure or where temporary connectivity is needed, *e.g.* emergency situations, disaster relief scenarios, and military applications.

The changing topology of MANETs and use of the wireless medium justify the need for different routing protocols than those developed for wired networks or multi-cell environments. Various routing protocols have been proposed in the IETF MANET working group to address the problem of decentralized routing. In this document, we consider the reactive routing protocol known as Dynamic Source Routing (DSR), one of the key protocols proposed for the use in a MANET environment.

What are the main features of DSR?

The Dynamic Source Routing (DSR) protocol was designed especially for MANET applications. Its main feature is that everydata packet follows the source route stored in its header. This route gives the address of each node through which the packet should be forwarded in order to reach its final destination. Each node on the path has a routing role and must transmit the packet to the next hop identified in the source route.

Each node maintains a Route Cache in which it stores every source route it has learned. When a node needs to send a data packet, it checks first its route cache for a source route to the destination. If no route is found, it attempts to find one using the route discovery mechanism.

A monitoring mechanism, called route maintenance, is used in each operation along a route. This mechanism checks the validity of each route used.

How does the DSR's route discovery work?

If node S wants to communicate with node D, it needs to find a route on demand by using the route discovery mechanism. Node S broadcasts a Route Request packet in the network. This Route Request contains the address of the initiator, the address of the target, a field sequence number (set by the initiator and used to identify the request), and a route record. The latter is the field where a record of the sequence of hops taken by the Route Request is accumulated dynamically. Each node in the network maintains a table in order to detect a duplicate Route Request packet received.

A node propagates the Route Request if it is not the target and if it is the first time it receives this packet. The first node receiving this Request that has a valid route in its route cache for node D initiates a Route Reply packet back to node S. This Route Reply contains the list of nodes along the path from node S to node D. The first part is the information gathered along the path of the Route Request (that is, from node S to the node replying); the rest of the list is the information found in the route cache of the replying node. Moreover, it may occur that destination node D itself receives a Route Request packet, *e.g.* no node along the way before node D has an accurate route from node S to node D in its route cache. In this case, node D sends a Route Reply packet containing the path just created dynamically from source S to destination D, *i.e.*, the path traversed by the first Route Request packet received by node D. This path is the minimum delay route from node S to node D. Node D discards all Route Request packets corresponding to the same route discovery process after the arriving of the first one.

How does DSR route maintenance work?

The route maintenance mechanism ensures that the paths stored in the route cache are valid. If the data link layer of a node detects a transmission error, the node creates a Route Error packet and transmits it to the original sender of the data packet. This Route Error packet indicates which link is “broken”, *i.e.*, the node that detected the error and the node it was trying to reach. When a node receives a Route Error packet, it removes the link in error from its route cache and for each route containing this link, truncates the route from the hop before the broken link.

In order to have feedback on the status of each hop, several acknowledgement mechanisms may be used, *e.g.* acknowledgement at the MAC layer level, request of an explicit acknowledgement from the next-hop receiver in the data packet header, or passive acknowledgement (that is, a node overhears the next node forwarding its packet).

Why a DSR model for OPNET?

Today, many industries are interested in Mobile Ad Hoc Networks, and one of the most important challenges to take up in this kind of network, where there is no fixed topology, is the routing issue.

Many routing protocols have been submitted, such as AODV, ZRP, and DSR. One of the present goals is to compare their performances in order to figure out which ones are the more suitable for MANET applications, or at least for particular MANET applications.

Some comparisons have been done, especially in universities, with free simulation tools such as NS-2 or GloMoSim, giving a preliminary idea of protocol performances in a wireless situation. At NIST, our goal is to help industries in the standardization of such protocols, thus providing our support to the study of routing protocol for MANET. That is why the Wireless Communications Technologies Group decided to evaluate some of these routing protocols from another point of view by using the OPNET software.

In this report we explain our modelling choices for one of these protocols, DSR, comparing with its latest specification (04) that we have tried to follow as best as possible.

The DSR node model

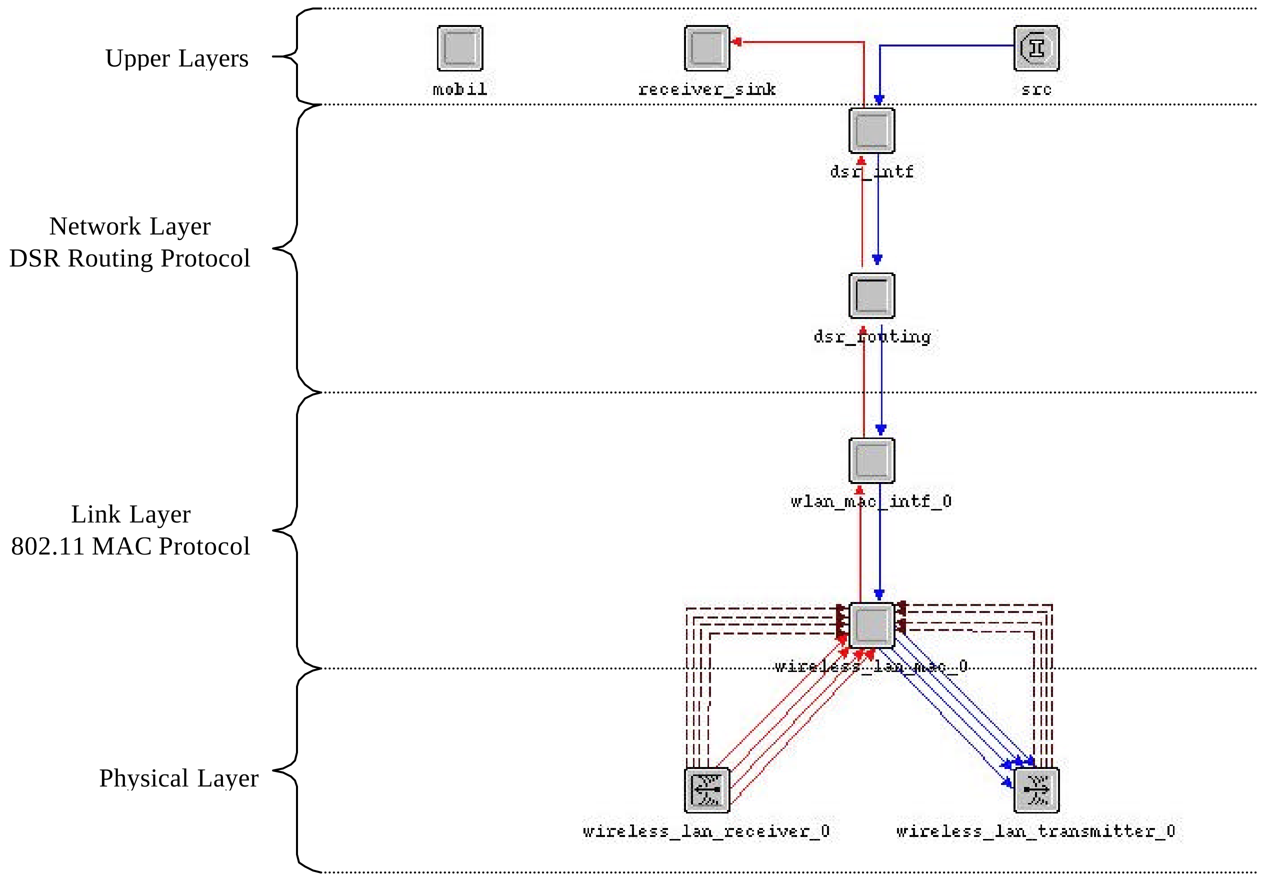
How does the modeling work under OPNET?

To explain quickly, the modeling under OPNET is divided into 3 levels. The upper level is the network level, and consists in the building of the network topology. The second level is the description of each node that composed this network topology. Thus a node model, which is like a stack of processes, must be provided for each kind of node. The third and last level is the description of each of these processes that composed a node model. These processes models are designed thought a state machine based on C code.

Our DSR node model

Actually, our DSR node model is like a stack of processes where each process (or group of processes) is a layer of the OSI communication protocol model. Here is the DSR node model scheme:

The DSR Node Model



Here is a detailed description of each layer of the DSR node model:

- The physical layer is composed of a transmitter and a receiver. Each of these blocks is not really an OPNET process, but define every C source code that is used in the pipeline wireless communication mechanism (see the OPNET documentation). All of these source codes have been developed by OPNET. Actually, both the physical and the link layer of this DSR node model come from the Wireless_Lan model developed by OPNET.

- The link layer used in the model is the OPNET 802.11 model. Note that some small modifications have been done in order to link this MAC layer with our DSR routing layer (sending of acknowledgement and error messages, addition of the promiscuous mode, and fixing of small bugs relative of the nodes mobility). We choose to use 802.11 since this protocol is presently in fashion in the MANET community, and seems to be the protocol that will be used in this kind of network. This protocol is divided in two processes. The first one (wireless_lan_mac_0) is the 802.11 protocol itself, and the second one (wlan_mac_intf_0) is an interface with the upper layer.

- The network layer is the core of our node model since it contains our DSR routing process model. As the 802.11 layer, it is divided in two processes. The first one is the DSR routing module, and the second one an interface with the upper layer that chooses for instance a random destination address for each data packet that must be transmitted on the network. Thus this report will focus in its next and last section on the first process containing the DSR routing protocol.

- The upper layers are mainly composed of two processes. The source is an OPNET process that generates data packet traffic. The receiver is a sink that just destroys the packet after the reception and processing completion.

- Note that we place in the upper layers another process model, called mobil. This one is in charge of the node mobility. We developed two special mobility models. The first one, called billiard_mobility, emulates the trajectory on a billiard table. That is, each node chooses a random direction that it follows at a constant speed until it reaches the network grid boundaries, where it rebounds by choosing a new random direction. The second one is the waypoint mobility, where a node computes a random destination point in the network space, then moves to this point at a constant speed. Upon reaching this point, the node pauses for a wait timer and then computes a new destination.

The DSR routing process model

This last section will focus on the DSR routing model that we have developed at NIST. Every mechanism that has been implemented will be listed and compared to the DSR specification. Note at this point that we developed our model with the draft-ietf-manet-dsr-03.txt. However, the fourth version of this draft recently appeared on the Internet, and we decided to use this latest version in our comparison. In the second part of this section the finite state machine of our DSR model will be described. Note that you will find in the appendix technical details of the model, such as the functions, the variables, and packet format descriptions.

The NIST DSR protocol implementation.

This part describes how our DSR protocol is implemented under OPNET in comparison with the protocol specification. First, we focus on the route discovery mechanisms, then on the route maintenance mechanisms. To conclude, we explain our route cache strategy.

Route Discovery mechanisms

- **Route Request mechanisms**

To avoid the propagation of many redundant copies of the Route Request packet, the originator of the packet initially limits the Request's propagation to one hop. This limitation allows the node to check whether one of its neighbors has a route to the destination or if the target is within transmission range. For this purpose, the node sends a Non-propagating Route Request.

If no Route Reply is received after a short timeout period, a "classic" or propagating Route Request is sent, and the hop limit is set to a predefined value (7 in our model, *i.e.* the maximum length of a route).

Finally, as long as no Route Reply is received, a new propagating route request is sent, but with an exponential increasing back-off time in order to reduce overhead. This is particularly useful and efficient when the network is partitioned, since two nodes in two different elements of the partition cannot exchange packets and it is useless to look for a route between them.

- **Route Request from error mechanisms**

The route request used after an Error Packet reception is quite different from the one described above. Indeed, since an error has been just detected, the neighbor nodes cannot have a valid route to the destination in their route cache. Consequently it is useless to consult them via a non-propagating Route Request, and a propagating Route Request is sent directly.

Moreover in the latest DSR specification, when a node receives such an Error packet, it should piggyback in its reacting Route Request the Error message. Actually this new mechanism avoids the situation in which a node, which did not hear the Error packet, replies to the Request by using a route in its cache containing the broken link. This mechanism seems to be very interesting, but we did not have the time to implement it yet.

- **Delayed Route Reply Mechanism**

The main objective of this mechanism is to avoid Route Reply storms, which happen when non-destination nodes use their ability to reply to a Route Request using their Route Cache information. For instance, when a node broadcasts a Route Request, each of its neighbours receives it at approximately the same time. Unless something is done to prevent it, all the neighbors able to reply using their Route Cache will send their Route Replies at about the same time. This will result in the so-called "Route Reply storm", that is, packet collisions and node congestion.

To avoid this phenomenon, each Route Reply is delayed following the formula:

$$Delay = DELAY\_PER\_HOP \times (NB\_hops - 1 + R(0,1))$$

Where: $DELAY\_PER\_HOP$ *is a constant (* 10^{-5} *s in our model),*

NB_hops *is the number of hops of the returned route,* $R(0,1)$ *is a random_number_between_0_and_1).*

Using this delay, the nodes do not send their Route Reply at the same time. Moreover, the first Reply to be sent contains the shortest route to the desired destination in harmony with the goal of DSR to find the shortest route.

Note that in order to give priority to a Route Reply sent directly by the target of the Route Request, the delay at the target is 0.

- **Promiscuous mode**

In this part we are considering only the promiscuous mode used in the route discovery mechanism.

The promiscuous mode gives a node the ability to change its schedule because of a certain event. For example, since the first Route Reply that is sent on the network contains the shortest route to the destination, each node which hears this packet and which is waiting for sending its own Route Reply, knows that its own Route Reply is not useful any more. Thus it cancels its scheduled Reply, which results in less traffic in the network. Note at this point that in the latest DSR specification, this promiscuous mode mechanism should be active on the first data packet sent after the Route Reply reception instead of the first Route Reply packet to be sent. However we did not have enough time to think about this new issue and to implement it.

- **Loop free**

As explained in the specification, the model ensures that each path is loop free. For instance, it removes every loop from the route constructs by a relay node using its route cache, and it checks that the relay node is still in the path after the processing. It also ensures that a Request Packet is forwarded one and only one time by a node.

- **Non-implemented mechanisms**

The first mechanism that is not implemented in our model and that is described in the latest DSR specification is the jitter delay on the Route Request and on the Route Reply packets transmission. Actually this idea appears only in the latest DSR specification and we did not have enough time to think about this issue. At first view, we believe that the 802.11 backoff and defer mechanisms provide almost the same behavior by waiting a short time before transmitting the packet on the physical layer.

The second thing to notice is that we did not implement any time-out for a data packet that is waiting for a route discovery completion. We believe that this mechanism only hides the real performances of DSR by reducing artificially the delivery delay, without any improvement in the efficiency for example. Since we want to evaluate the real performances of DSR, it was not our interest to implement this mechanism.

Finally, another choice that we made is to wait for the Reply Packet when a node is in the route discovery process, and to transmit the corresponding data packets stored in memory only when this reply is received. In the specification it is advised to send the packet as soon as any packet providing a route to the destination is received. This mechanism is interesting, even if the route may be less safe that the one returned in a real reply. However, since we did not implement every promiscuous mode allowing the learning of new route, we believe that it is not really useful and does affect the protocol performances.

Route Maintenance mechanisms

- **Acknowledgement mechanism**

In our current OPNET DSR model, we use the IEEE 802.11 MAC layer as link layer. Thus this layer provides the acknowledgement and error messages required by the routing layer.

It is important to note that our present model is not able to transmit any explicit network layer acknowledgement, although this mechanism is almost implemented. Actually, our model does not have any retransmission buffer, but is able to detect via a timer, that no acknowledgement message has been received for a previously transmitted data packet. This mechanism is presently used to ensure that the 802.11 layer works properly, that is, sends either an acknowledgement or an error packet for every data packet emitted. Obviously, it will be also the basis for a future explicit DSR acknowledgement implementation.

- **Promiscuous mode**

The first application of the promiscuous mode in the route maintenance mechanism is to update the node’s route cache when it hears any packet. For example, when a node hears a Route Error packet where a broken link is specified, it removes the link from its Route Cache even if it was not the target of the Route Error packet. Moreover in the specification a node should update its Route Cache by using the path contained in any packet that it is able to overhear. But, due to our Route Cache strategy, as explained in the next sub-section, we did not implement these promiscuous mechanisms.

Another application of the promiscuous mode for the route maintenance is to use that “hearing” feature in order to “reflect” shorter routes. For example, if a relay or a destination node determines that a previous relay on the path is unnecessary because of topology changes, it sends an unsolicited Route Reply packet to the original sender of the Data packet in order to reflect the shorter route found. Note also, that in the specification a relay node that is sending a Gratuitous Route Reply should wait for the second packet that follow the “normal” path before forwarding it on the next hop. We decided to use another method, by sending directly the first received packet, since it can only provide a better efficiency (we are sure that the node has received the first packet, but we are not sure that it will receive the second one). To implement this mechanism we simply add an ID to each data packet in order to be able to determine if a packet has already been received and forwarded by a node

Route Cache Strategies

One route for each destination

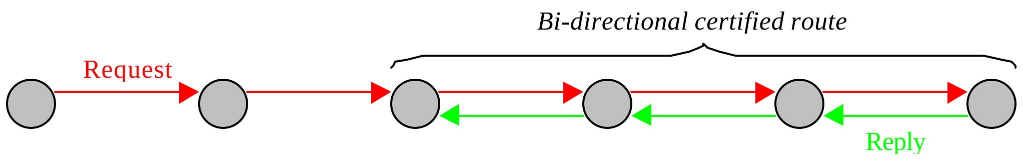
As we show in the variable descriptions part located in the appendix, our route cache contains only one route for each destination. We made this choice since our first goal when we designed the model was to evaluate the basic DSR mechanism's behavior and performance, and not the influence of its different route cache strategies.

Moreover, we believe that the real power and interest of a multiple route cache is located in its ability to be used in order to choose a route considering other metrics than the shortest path (such as the power requirement, the traffic load…). DSR does not collect such metrics presently, which explains why we do not choose this solution. However, we already did some research about this issue that resulted in our Cost Adaptive Mechanism (CAM). Now our new challenge will be to find a way in order to choose the "best" metric in each given situation.

Bi-directional links

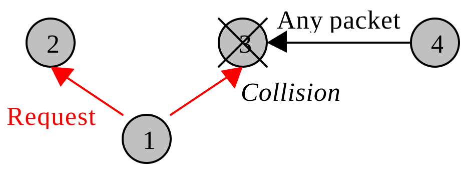
The first consequence of using the 802.11 MAC layer is that links between nodes must be bi-directional. Thus we loose one of the main behavior of DSR: its ability to manage unidirectional links.

The main consequence is that when a node receives a Reply packet, we can only update its route cache with the path part located between itself and the reply source node, since only this portion is "validated" bi-directional. The following diagram explain that idea:

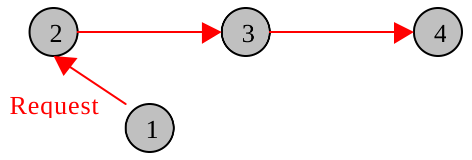


The other reason for using only this part of the path for the route cache updating is that we have noticed that sometimes the Route Request packet does not follow the shortest path as described in the specification. Actually, this problem is not addressed in the DSR specification, and the following diagrams describe clearly when it can occur.

1) Node 1 broadcasts its Route request but node 3 cannot hear it due to a collision (packet coming from node 4 for example)

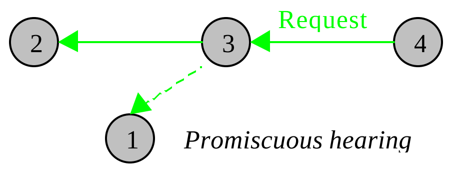


2) Thus the Request that will be received by 4 will have taken the following route, which is not the shortest one.

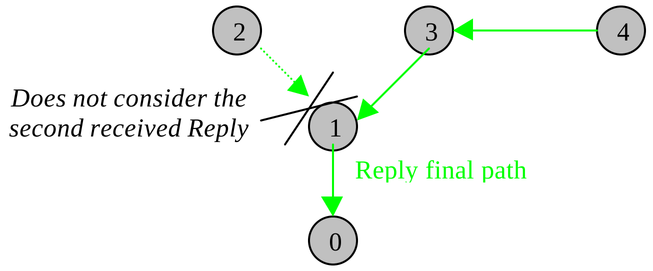


Our solution to fix this problem was to implement a mechanism similar to the gratuitous Route Reply used on the data packet. This mechanism is described in the diagrams following this paragraph. Actually, when a node detects a shortest path when it receives a reply packet in promiscuous mode, it changes the route in the Reply header and broadcasts it. Obviously when it receives the second Reply that has followed the "normal" route, it will destroy it. Thus by using only the bi-directional confirmed part of the route to update the route cache of each node on the path, we also ensure that we use the shortest routes to the destination.

3) Node 1 will receive the Reply in promiscuous mode before the path contained in the packet schedules it.



4) Node 1 will consider this Reply packet heard in promiscuous mode, modify its route, eventually (if it is not the Reply Destination) forward it the next node on the path (here 0), and forget the next Reply packet corresponding to the same request coming from node 2.



<u>Other non-implemented mechanisms</u>

- **No IP implementation**

We did not implement any IP idea, concept or mechanism in our model. The reason is that our goal is to evaluate the DSR concept and mechanism, and not its IP implementation.

- **No Piggybacking**

Our model does not implement any Piggybacking mechanism. We made this choice when we decided to use DSR above the 802.11 MAC layer. Indeed, the main objective of the Piggybacking mechanism in the third DSR specification is to allow for non bi-directional links, which is incompatible with 802.11.

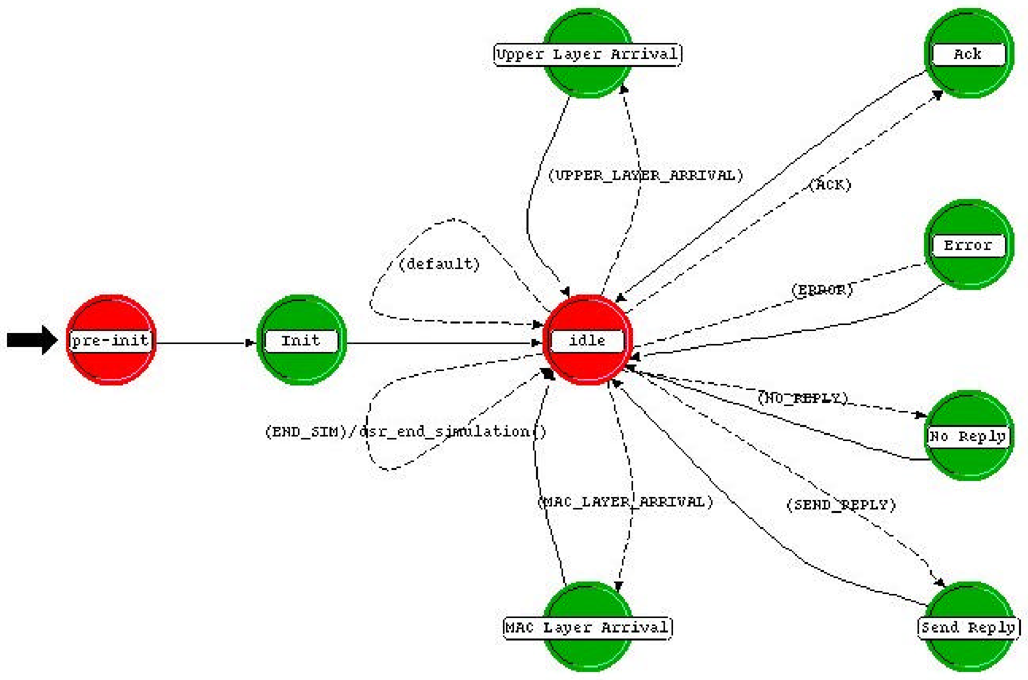
Nevertheless, in the last and all new specifications of DSR the piggybacking is also used in a very interesting new mechanism. We already discussed this mechanism, which consists of piggybacking in a Request packet reacting to an Error packet this Error message. As we said, we are thinking about implementing this mechanism in the future.

- **No Salvaging**

The salvaging mechanism is new in the latest DSR specification, thus we did not have enough time to implement it. However during our model development, we thought of and even tried a similar mechanism, which consists of re-routing a data packet when a node detects a broken link. We faced many problems, and we believe that some more work need to be done on the specification in order to implement this mechanism easily and efficiently.

The DSR process model state machine

In this part the DSR process machine will be described. First, the following diagram shows what this finite state machine look like under the OPNET modeler, then an explanation will describe the role of each state.



Pre-Init State:
This state pre-initializes the DSR process model by managing the DSR address of the current node, and by checking that this address is valid within the network.

Init State:
This state initializes every variable, statistic, table, and user parameter that is used by the DSR process model.

Idle State:
This is the default state where the process waits for an event.

Upper Layer Arrival State:
This state handles every packet generated by the upper layer that the DSR protocol must carry to a given destination.

MAC Layer Arrival State:
This state handles every packet received from the 802.11 link layer. It processes the packet differently depending on its type: request, reply, data, or error.

Send Reply:
This state is “called” when the current node must send a scheduled reply from relay. Thus this state just sends this scheduled Reply packet.

No Reply:
This state is “called” when the timeout associated with a Request packet transmitted by the node expires. That means that no Reply packet has been received, thus that the previous step of the route discovery failed, and that a new Request packet must be generated by the node. This state does all these operations.

Ack State:
This state handles every acknowledgement coming from the 802.11 link layer. An acknowledgement confirms that the current link used is valid, thus allowing the node to send other Data packets through this link. At the same time, it resets the error mechanism (no ack reception) that checks either if an explicit DSR layer acknowledgement was received (will be implemented) or if the MAC layer is working (currently implemented)

Error State:
This state occurs when an error is detected by the DSR layer error mechanism or is received from the 802.11 MAC layer. The link, which was used for the data packet, is declared “broken”, resulting in the route cache updating and in an Error packet transmission.

Note that presently every error and acknowledgement message is waited from the MAC layer (the explicit layer acknowledgements will be implemented later). However the DSR error mechanism is already implemented, which means two things: On the one hand, when an error message is received from the MAC layer, this mechanism must be reset to avoid useless DSR layer error detections. On the other hand, if a DSR layer error is nevertheless detected, it means that neither error nor acknowledgement message was received from the MAC layer. In other terms it means that the MAC layer is not working. That is why the simulation is stopped in this case.

Conclusion

This report explains how we have implemented the DSR routing protocol under OPNET. This model was build with the third DSR draft, and since the fourth one just appeared, we have compared our model to this last specification.

Our model main goal is the evaluation of the routing protocols for MANET applications. Thus, we made some design choices sometimes differing slightly from the specification in this way. Moreover, other new points that have just appeared in the last specification have not been implemented due to an obvious lack of time. We intend to add some of these new mechanisms in the near future.

Nevertheless, since we believe that our model is good enough to be useful to the OPNET community, we decided to provide it right now.